Flash安全

过去、现在、和将来

Haifei Li, security researcher
Haifei_Li@McAfee.com

McAfee
An Intel Company

SAFE NEVER SLEEPS.

这家伙是谁?

- 安全发烧友，主要关注应用程序级别的漏洞发掘和利用.
- 2005-2008年期间活跃于国内安全圈，曾用id “cocoruder”，有印象?



From the Internet. For the Internet.

首页 | 焦点原创 | 安全文摘 | 安全工具 | 安全漏洞 | 焦点项目 | 焦点论坛

文章分类

- 专题文章
- 漏洞分析
- 安全配置
- 黑客教学
- 编程技术
- 工具介绍
- 火墙技术
- 入侵检测
- 破解专题
- **焦点公告** <<
- 焦点峰会

文章推荐

阿里巴巴支付宝远程代码执行漏洞-0DAY

创建时间：2007-02-07
文章属性：原创
文章提交：cocoruder (frankruder_at_hotmail.com)

阿里巴巴支付宝远程代码执行漏洞

by cocoruder(frankruder_at_hotmail.com)
http://ruder.cdut.net

Summary:

支付宝是由阿里巴巴公司出品的在线支付解决方案，在中国拥有众多用户，更多信息请参考：

https://www.alipay.com

在支付宝密码输入控件中存在一个远程代码执行漏洞，远程攻击者可利用此漏洞在被攻击者系统.

这家伙是谁？(继续)

- 在 华为安全实验室 收到第一次工资
- 伴随 Fortinet公司 一起成长（5年）
- 曾为 微软公司 工作
- 目前是 迈克菲(McAfee)实验室 的一名成员，和一些很棒的朋友一起工作
- 居住在加拿大温哥华 – 地球上最漂亮的城市
  - 如果你来温哥华玩，请务必找我☺

改变: 漏洞研究之我见

- 发现新漏洞的困难度 --
  - 自动化发掘(fuzzing)技术和平台已经很成熟了
  - 我们日常使用的软件越来越复杂
    - 更多的功能 = 更多的漏洞
  - 用于fuzzing的硬件（PC/server）越来越便宜

  - 动机
    - 我们有很多漏洞发掘的现金奖励计划! (ZDI, iDefense, Google, etc).
    - 安全公司愿意投入漏洞发掘以提升自己的名声和安全检测产品性能。

改变: 漏洞研究之我见

- 漏洞利用的困难度 ++
  - ASLR + DEP，新版“海尔兄弟”
  - 越来越多的程序武装“沙盒”技术
  - 任何事情都“随机化”(不可预测性)
    - 漏洞利用者们喜爱“可预测性”，而不单单是亚洲人☺
  - 动机
    - 我们并没有一个奖励计划来激励开发新的漏洞利用技术
    - 而不仅是被邀请去安全会议 :P

Agenda

1	回顾Flash威胁
2	Flash ActionScript 3 架构
3	Flash JIT和减轻技术
4	案例: 利用Tamarin源代码来分析Flash漏洞
5	未来Flash安全的思考

历史

- Flash Player的安全记录并不光彩

历史

- 2011年3月份为我的CanSecWest演讲做的搜索

Search Results

There are 147 CVE entries or candidates that match your search.	CVE version: 20061101

Name	Description
CVE-2011-0608	Adobe Flash Player before 10.2.152.26 allows attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors, a different vulnerability than CVE-2011-0559, CVE-2011-0560, CVE-2011-0561, CVE-2011-0571, CVE-2011-0572, CVE-2011-0573, CVE-2011-0574, CVE-2011-0578, and CVE-2011-0607.
CVE-2011-0607	Adobe Flash Player before 10.2.152.26 allows attackers to execute arbitrary code or cause a denial of service (memory corruption) via unspecified vectors, a different vulnerability than CVE-2011-0559, CVE-2011-0560, CVE-2011-0561, CVE-2011-0571, CVE-2011-0572, CVE-2011-0573, CVE-2011-0574, CVE-2011-0578, and CVE-2011-0608.

- 269 - 147 = 122 个漏洞增加在20个月的时间跨度

历史

- 最初的Flash Player漏洞可以追踪到2002年

CVE-2002-0476

Summary: Standalone Macromedia Flash Player 5.0 allows remote attackers to save arbitrary files
"save" FSCommand.

Published: 08/12/2002

CVSS Severity: 5.0 (MEDIUM)

CVE-2002-0477

Summary: Standalone Macromedia Flash Player 5.0 before 5,0,30,2 allows remote attackers to exe
FSCommand.

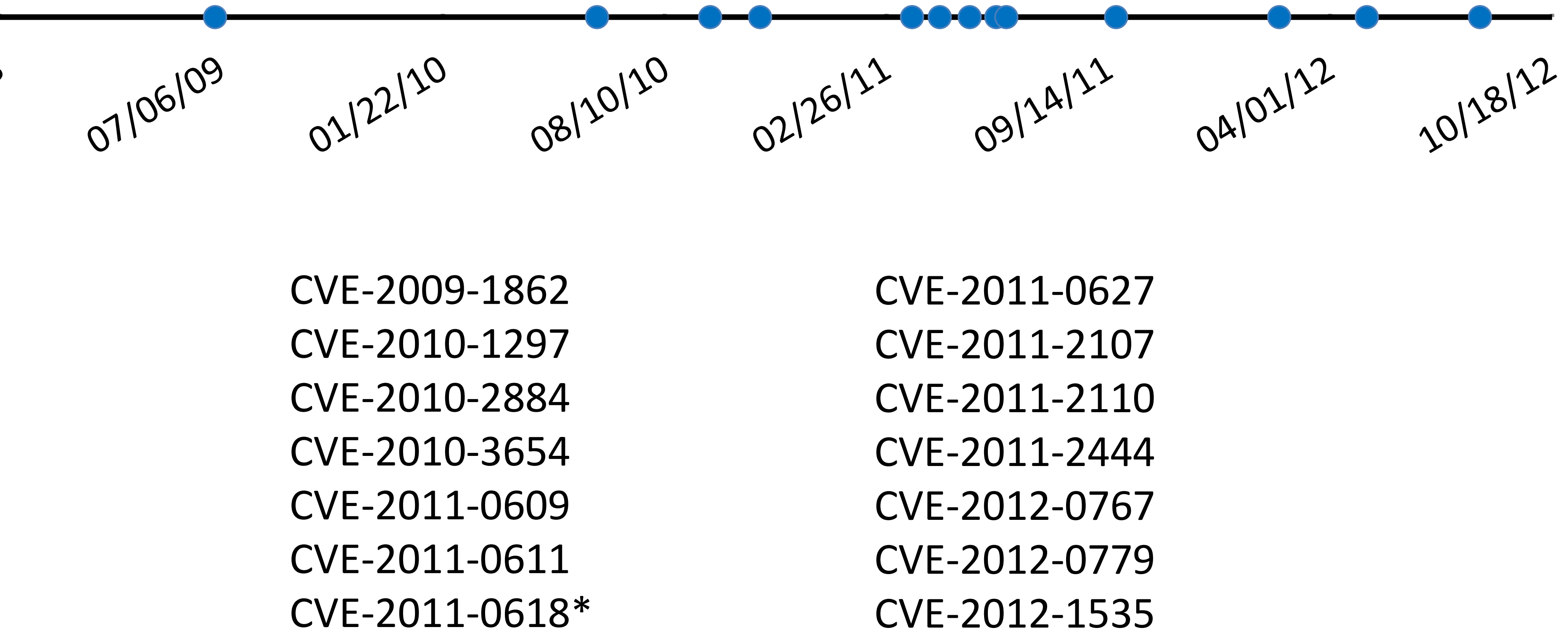
Published: 08/12/2002

CVSS Severity: 7.5 (HIGH)

<< < 3 4 5 6 7 8 9 10 11 12 13

Flash 0days

- 一个从没有结束的故事



CVE-2009-1862
CVE-2010-1297
CVE-2010-2884
CVE-2010-3654
CVE-2011-0609
CVE-2011-0611
CVE-2011-0618*

CVE-2011-0627
CVE-2011-2107
CVE-2011-2110
CVE-2011-2444
CVE-2012-0767
CVE-2012-0779
CVE-2012-1535

*According to a research paper, CVE-2011-0618 was not detected while it actually was a 0day vulnerability.

演进：基于Flash的漏洞利用技术

- 通过重写AS3_argmask攻破AS验证器, Mark Dowd, 2008年4月
  - 很重要的研究，因为其揭示了ActionScript/AVM的执行过程之细节
  - 已修复
- ActionScript 堆喷射, 至少在2009年早些时候即发现被利用在exploit中
  - 通用堆喷射技巧
  - 仍然可用

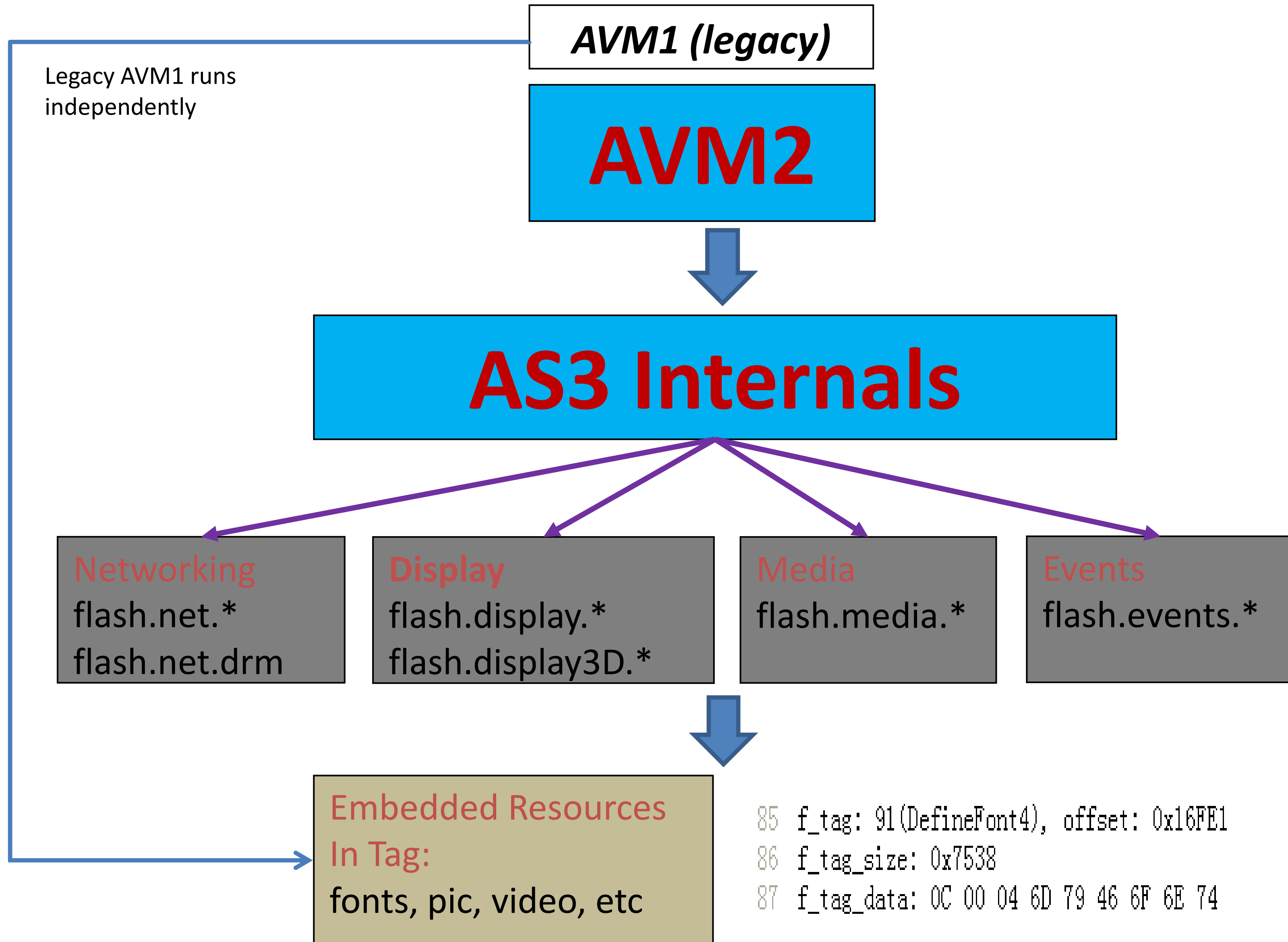
演进：基于Flash的漏洞利用技术

- JIT喷射, Dion Blazakis, 2010年2月
  - 意义重大的通用漏洞利用技术
  - 已修复（将在稍后讨论细节）
- AS3类型混乱利用, Haifei Li, 2011年3月
  - 只针对于此类漏洞类型，只能用于类型混乱漏洞
  - 仍然可用
- AS3本地API信息泄露, Fermin J. Serna, 2012年4月
  - 高度漏洞特定

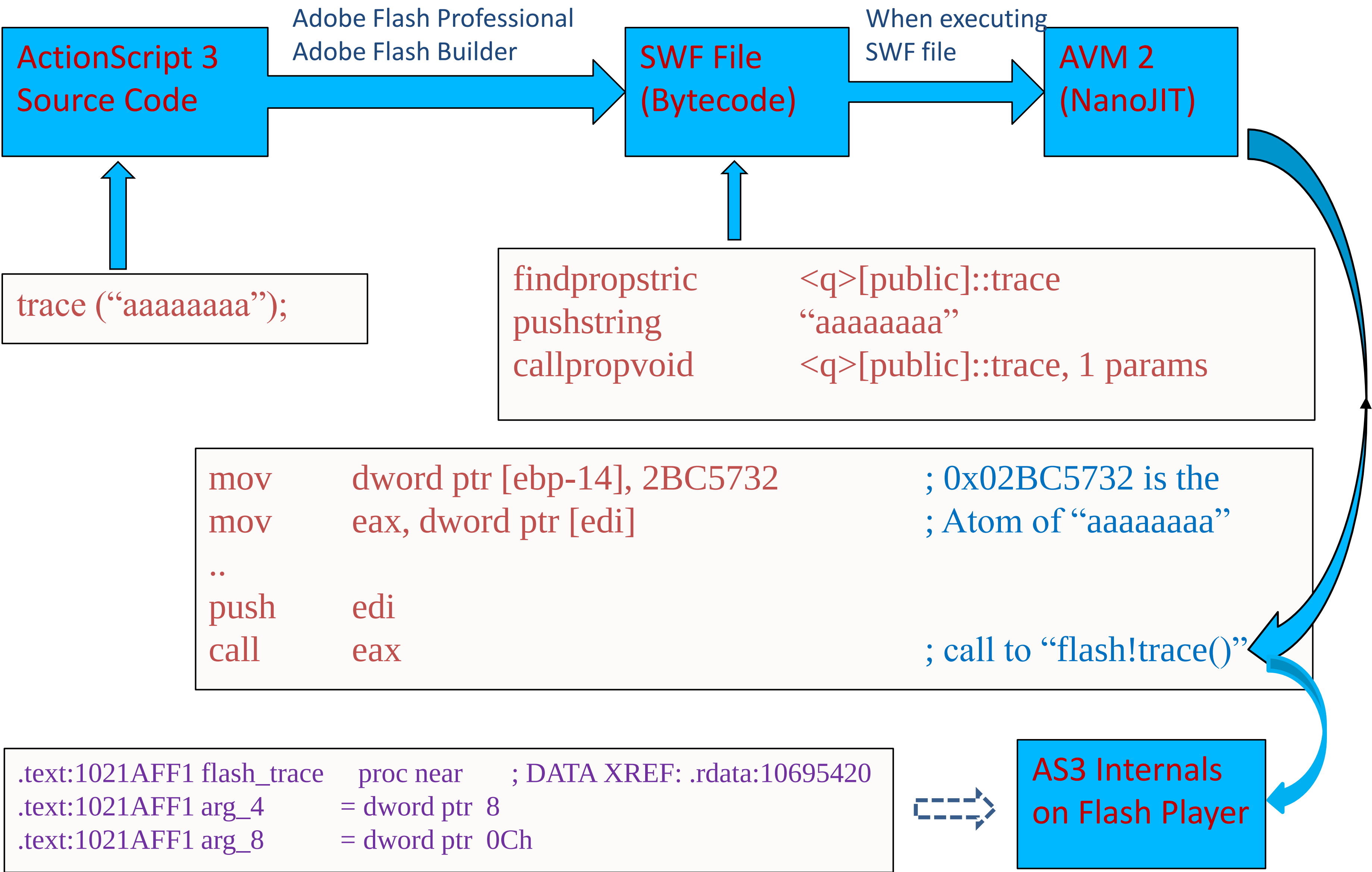
Agenda

1	回顾Flash威胁
2	Flash ActionScript 3 架构
3	Flash JIT和减轻技术
4	案例: 利用Tamarin源代码来分析Flash漏洞
5	未来Flash安全的思考

Flash Player架构



AS3执行过程



ActionScript 3
Source Code
Adobe Flash Professional
Adobe Flash Builder
SWF File
(Bytecode)
When executing
SWF file
AVM 2
(NanoJIT)
trace (“aaaaaaaa”);
findpropstric <q>[public]::trace
pushstring “aaaaaaaa”
callpropvoid <q>[public]::trace, 1 params
mov dword ptr [ebp-14], 2BC5732 ; 0x02BC5732 is the
mov eax, dword ptr [edi] ; Atom of “aaaaaaaa”
..
push edi
call eax ; call to “flash!trace()”
.text:1021AFF1 flash_trace proc near ; DATA XREF: .rdata:10695420
.text:1021AFF1 arg_4 = dword ptr 8
.text:1021AFF1 arg_8 = dword ptr 0Ch
AS3 Internals
on Flash Player

简单来说..

AVM2

+

AS3 Native APIs

=

Flash

- AVM2是开源的，参见Tamarin项目，它只是个语言解析器
- AS3本地API负责实现Flash的真正功能，它不是开源的
  - 因此如何检查这些API的安全性是一个挑战

Agenda

1	回顾Flash威胁
2	Flash ActionScript 3 架构
3	Flash JIT和减轻技术
4	案例: 利用Tamarin源代码来分析Flash漏洞
5	未来Flash安全的思考

JIT 喷射 / “风水”

- 最初由*Dion Blazakis* 在2010年2月的BlackHat DC 安全会议上发布
- 03396E6**A** (normal) vs 03396E6**B** (shellcode)

03396E6A	35 9090903C	xor	eax, 3C909090
03396E6F	35 9090903C	xor	eax, 3C909090
03396E74	35 9090903C	xor	eax, 3C909090
03396E79	35 9090903C	xor	eax, 3C909090
03396E7E	35 9090903C	xor	eax, 3C909090
03396E83	35 9090903C	xor	eax, 3C909090
03396E88	35 9090903C	xor	eax, 3C909090
03396E8D	35 9090903C	xor	eax, 3C909090
03396E92	35 9090903C	xor	eax, 3C909090
03396E97	35 9090903C	xor	eax, 3C909090
03396E9C	35 9090903C	xor	eax, 3C909090
03396EA1	35 9090903C	xor	eax, 3C909090
03396EA6	83EC 04	sub	esp, 4
03396EA9	50	push	eax
03396EAA	B9 00200003	mov	ecx, 3002000
03396EAF	E8 6CE40C0D	call	Flash11c.10465320

03396E6B	90	nop	
03396E6C	90	nop	
03396E6D	90	nop	
03396E6E	3C 35	cmp	al, 35
03396E70	90	nop	
03396E71	90	nop	
03396E72	90	nop	
03396E73	3C 35	cmp	al, 35
03396E75	90	nop	
03396E76	90	nop	
03396E77	90	nop	
03396E78	3C 35	cmp	al, 35
03396E7A	90	nop	
03396E7B	90	nop	
03396E7C	90	nop	
03396E7D	3C 35	cmp	al, 35

JIT 喷射 / “风水”

然而，你怎么知道你将要跳到 03396E6**B** 去，而不是 03396E6**A**？

JIT 喷射 / “风水”

03396E04	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90
03396E14	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C
03396E24	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35
03396E34	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90
03396E44	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90
03396E54	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90
03396E64	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C
03396E74	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35
03396E84	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90
03396E94	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90
03396EA4	90	3C	83	EC	04	50	B9	00	20	00	03	E8	6C	E4	0C	0D
03396EB4	83	C4	04	8B	4D	F0	89	0D	28	20	00	03	8B	E5	5D	C3
03396EC4	00	00	00	00	7C	5F	39	03	EC	7F	39	03	00	00	00	00
03396ED4	0C	7E	39	03	00	00	55	8B	EC	83	EC	28	89	5D	FC	8B
03396EE4	5D	08	8D	45	F0	8B	0D	28	20	00	03	89	5D	F4	89	4D
03396EF4	F0	89	05	28	20	00	03	8B	0D	1C	20	00	03	3B	C1	73
03396F04	07	8B	CB	E8	14	13	0D	0D	83	EC	04	6A	00	68	0C	0A
03396F14	02	03	53	E8	54	10	10	0D	83	C4	10	8B	50	08	8B	4A
03396F24	68	8D	55	E8	89	45	E8	C7	45	EC	CA	EB	19	03	8B	41
03396F34	04	83	EC	04	52	6A	01	51	FF	D0	83	C4	10	83	EC	04
03396F44	68	90	90	90	3C	B9	00	20	00	03	E8	1D	E4	0C	0D	83
03396F54	C4	04	8B	CB	8B	5D	FC	68	90	90	90	3C	50	E8	DA	98
03396F64	10	0D	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90
03396F74	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C
03396F84	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35
03396F94	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90
03396FA4	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90
03396FB4	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90
03396FC4	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C

JIT func2

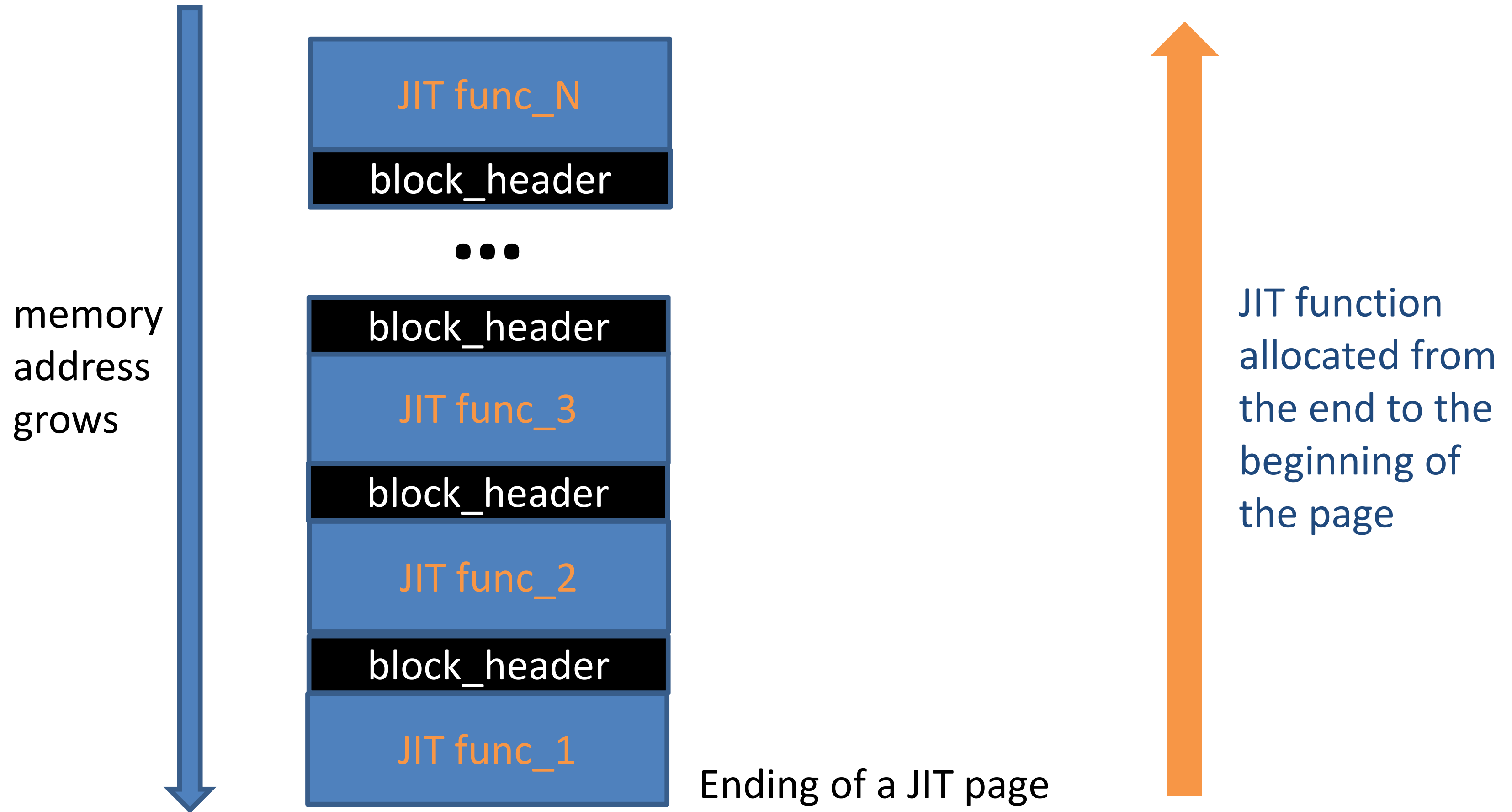
Padding + mem_header

JIT func1

JIT 喷射 / “风水”

- **事实:**
  - *mem_header* 结构固定长 0x14 字节
  - 当地址值可以被4整除时，*Padding* 为空
  - JIT函数的长度我们是可以控制的（通过控制AS函数的bytecode数目)
  - JIT分配依照反内存规则（从高内存向低内存），并且是连续的
- **结论:**
  - 通过在JIT页内存上喷射 (或者, “风水”)很多的精心定制的函数，所有的一切都是可控的
  - 早些被广泛使用的“JIT spraying Xor attacks”只是一个粗燥的应用

JIT 喷射 / “风水”



memory
address
grows
JIT func_N
block_header
...
block_header
JIT func_3
block_header
JIT func_2
block_header
JIT func_1
Ending of a JIT page
JIT function
allocated from
the end to the
beginning of
the page

JIT 加固技术

- 在很早的2010年10月就已经引入这个特性了

tamarin-redux - changeset - 5434:639c280f0cb6

summary | shortlog | changelog | graph | tags | files | changeset | raw | bz2 | zip | gz

Bug 595033 - nanojit: harden via random function alignment (wmaddox,nnethercote,edwsmith)

author	Rick Reitmaier <rreitmai@adobe.com>
	Thu Oct 14 18:54:07 2010 -0700 (at Thu Oct 14 18:54:07 2010 -0700)
changeset 5434	639c280f0cb6
parent 5433	a7313158044c
child 5435	6aea0e137aec
pushlog:	639c280f0cb6

JIT 加固技术

- 相关代码

```
     void Assembler::nativePageSetup()
     {
         NanoAssert(!_inExit);
         if (!_nIns)
             codeAlloc(codeStart, codeEnd, _nIns verbose_only(, codeBytes));
+
+        // add some random padding, so functions aren't predictably placed.
+        if (_config.harden_function_alignment)
+        {
+            int32_t pad = _noise->getValue(LARGEST_UNDERRUN_PROT);
+            underrunProtect(pad);
+            _nIns -= pad;
+            VMPI_memset(_nIns, INT3_OP, pad);
+            PERFM_NVPROF("hardening:func-align", pad);
+        }
     }
```

JIT 加固技术

- 然而，一直到2011年11月，才在发行的Flash Player 11.1.102.55 版本上正式启用这个特性
- 通过设置 配置标记变量 “*harden_nop_insertion*” 和 “*harden_function_alignment*”

```
else if (!VMPI_strcmp(arg, "-jitharden"))  {
    settings.njconfig.harden_nop_insertion = true;
    settings.njconfig.harden_function_alignment = true;
}
```

代码基址对齐随机化 Codebase Alignment Randomization

- 当向内存中写入JIT函数机器码时，在其前面插入随机个数的0xCC (INT3)
- 从最小的 **0** 到最大的 **0x19**

```
const int LARGEST_UNDERRUN_PROT = 32;

// add some random padding, so functions aren't predictably placed.
if (_config.harden_function_alignment)    {
    int32_t pad = _noise->getValue(LARGEST_UNDERRUN_PROT);
    underrunProtect(pad);
    _nIns -= pad;
    VMPI_memset(_nIns, INT3_OP, pad);
    PERFM_NVPROF("hardening:func-align", pad);
}
```

031A5E24	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35
031A5E34	90	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90
031A5E44	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90
031A5E54	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90	90
031A5E64	3C	83	EC	04	50	B9	00	20	E0	02	E8	FD	07	2C	0D	83
031A5E74	C4	04	8B	4D	F0	89	0D	28	20	E0	02	8B	E5	5D	C3	CC
031A5E84	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC
031A5E94	CC	CC	CC	CC	00	00	00	00	30	4F	1A	03	EC	6F	1A	03
031A5EA4	00	00	00	00	04	6E	1A	03	55	8B	EC	83	EC	28	89	5D
031A5EB4	FC	8B	5D	08	8D	45	F0	8B	0D	28	20	E0	02	89	5D	F4
031A5EC4	89	4D	F0	89	05	28	20	E0	02	8B	0D	1C	20	E0	02	3B
031A5ED4	C1	73	07	8B	CB	E8	92	36	2C	0D	83	EC	04	6A	00	68
031A5EE4	0C	0A	E2	02	53	E8	D2	32	2F	0D	83	C4	10	8B	50	08
031A5EF4	8B	4A	68	8D	55	E8	89	45	E8	C7	45	EC	12	EC	F9	02
031A5F04	8B	41	04	83	EC	04	52	6A	01	51	FF	D0	83	C4	10	83
031A5F14	EC	04	68	90	90	90	3C	B9	00	20	E0	02	E8	9B	07	2C
031A5F24	0D	83	C4	04	8B	CB	8B	5D	FC	68	90	90	90	3C	50	E8
031A5F34	D8	BA	2F	0D	35	90	90	90	3C	35	90	90	90	3C	35	90
031A5F44	90	90	3C	35	90	90	90	3C	35	90	90	90	3C	35	90	90

Next JIT func

Random 0xCC inserted

mem_header

Previous JIT func

指令对齐随机化 Instruction Alignment Randomization

- 在任意（随机选择的）**0x80-0x47F**的距离范围内，插入一个“空指令”
- 当前有 5 个“空指令”:
  - *8B C0 mov eax, eax*
  - *8B FF mov edi, edi*
  - *8B C9 mov ecx, ecx*
  - *8D 09 lea ecx, [ecx]*
  - *8D 24 24 lea esp, [esp]*

- 提示: 对用函数长度小于0x80字节的，并不会插入“空指令”

指令对齐随机化 Instruction Alignment Randomization

```
// if no codeList then we know priorIns and _nIns are on
   same page, otherwise make sure priorIns was not in the
   previous code block

if (!codeList || !codeList->isInBlock(priorIns))   {
   // sanity check
   NanoAssert(delta < VMPI_getVMPageSize());
   nopInsertTrigger -= (int32_t) delta;
   if (nopInsertTrigger < 0)  {
        nopInsertTrigger = noiseForNopInsertion(_noise);
        asm_insert_random_nop();
        PERFM_NVPROF("hardening:nop-insert", 1);
   }
}
```

033A703B	35 9090903C	xor	eax, 3C909090
033A7040	35 9090903C	xor	eax, 3C909090
033A7045	35 9090903C	xor	eax, 3C909090
033A704A	35 9090903C	xor	eax, 3C909090
033A704F	35 9090903C	xor	eax, 3C909090
033A7054	8D09	lea	ecx, dword ptr [ecx]
033A7056	35 9090903C	xor	eax, 3C909090
033A705B	35 9090903C	xor	eax, 3C909090
033A7060	35 9090903C	xor	eax, 3C909090
033A7065	35 9090903C	xor	eax, 3C909090
033A706A	35 9090903C	xor	eax, 3C909090

JIT 加固技术 – 效果

- 通过同时启用这两个“随机化”技术，基本上堵住了所有典型的 JIT喷射/ “风水”攻击
  - 内存地址再也不是可预测的了(代码基址对齐随机化).
  - 函数长度再也不是可控制的了(指令对齐随机化)
  - “空指令”所带来的字节会破坏你的JIT shellcode
  - 甚至，你无法猜测出任意一个JIT地址上的字节值

Agenda

1	回顾Flash威胁
2	Flash ActionScript 3 架构
3	Flash JIT和减轻技术
4	案例: 利用Tamarin源代码来分析Flash漏洞
5	未来Flash安全的思考

利用Tamarin源代码来分析漏洞成因

- Tamarin项目是开源的AS3(AVM2)执行过程
- 因此，对于基于AVM2的漏洞，我们可以利用源代码来定位到漏洞成因
- 从源代码级别，我们就可以完全理解漏洞的本质，以及如何来利用它
- 下面我们将用一个最近的实际例子来说明

Note: 作者发布了一个专注于Tamarin项目的研究，你可以在这里找到 http://recon.cx/2012/schedule/events/210.en.htm.

背景

- 2012年10月12日，有人公布了一个刚刚被修补的漏洞PoC (1-day)
- 10月15日，McAfee实验室发布了一个blog post 描述了我们对此的相应
- 此漏洞编号是CVE-2012-5271 (由Google报告), Adobe PSIRT已对此确认

分析

- 通过对原始样本文件和PoC文件的比较，发现只有1个opcode的参数被修改了。显然，问题出在AS3层面。

```
 constructor * <q>[public]::Evil=()(0 params, 0 optional)
[stack:3 locals:1 scope:8-8 flags:]
{
    00000) + 0:0 getlocal_0
    00001) + 1:0 constructsuper 0 params
    00002) + 0:0 getlocal_0
    00003) + 1:0 pushint 0
    00004) + 2:0 getlocal r0
    00005) + 3:0 getproperty <multi>{[private]"",[protected]"",[pack
    00006) + 3:0 coerce_a
    00007) + 3:0 callpropvoid <q>[public]::addFrameScript, 2 params
    00008) + 0:0 returnvoid
}
```

```
 constructor * <q>[public]::Evil=()(0 params,
[stack:3 locals:1 scope:8-8 flags:]
{
    00000) + 0:0 getlocal_0
    00001) + 1:0 constructsuper 0 params
    00002) + 0:0 getlocal_0
    00003) + 1:0 pushint 0
    00004) + 2:0 getlocal r0
    00005) + 3:0 declocal r10
    00006) + 3:0 coerce_a
    00007) + 3:0 callpropvoid <q>[public]::ad
    00008) + 0:0 returnvoid
}
```

深入分析

- declocal r10？ 为什么呢？
- 我们可以看见，“locals:1”意味着这个函数只有一个本地寄存器，因此只有r0可用。
- 似乎，有些东西“越过边界”了。。

深入分析

- 检视Tamarin源代码，在函数 Verifier::verifyBlock (代码文件 verifier.cpp)里，它读取并验证每个opcode的参数
  - http://hg.mozilla.org/tamarin-redux/file/f5191c18b0e4/core/Verifier.cpp
- 让我们定位到处理 declocal opcode的地方

```
case OP_inclocal:
case OP_declocal:
{
    Traits* retType = NUMBER_TYPE;
    bool already_coerced = false;
#ifdef VMCFG_FLOAT
    if(pool->hasFloatSupport())
    {
        FrameValue& v = checkLocal(imm30);
        BuiltinType bt = Traits::getBuiltinType(v.traits);
        if(bt == BUILTIN_none || bt == BUILTIN_any || bt == BUILTIN_object)
        {
            emitCoerceToNumeric(imm30);
            retType = OBJECT_TYPE;
            already_coerced = true;
        }
        else if(bt == BUILTIN_float || bt == BUILTIN_float4)
            retType = v.traits;
        else
            retType = NUMBER_TYPE;
    }
#endif // VMCFG_FLOAT
    if(!already_coerced)
        emitCoerce(retType, imm30);
    coder->write(state, pc, opcode, retType);
    break;
}
```

一些事实

- 对于正式发行的Flash Player来说，
  - 标记 *VMCFG_FLOAT* 并不会被设置
  - 标记 *already_coerced* 也不会被设置（为 *false*）

- 因此，对于上述的源代码来说。。

```
case OP_inclocal:
case OP_declocal:
{
        Traits* retType = NUMBER_TYPE;
        bool already_coerced = false;
#ifdef VMCFG_FLOAT
         if(pool->hasFloatSupport())
        {
                 FrameValue& v = checkLocal(imm30);
                BuiltinType bt = Traits::getBuiltinType(v.traits);
                if(bt == BUILTIN_none || bt == BUILTIN_any || bt == BUILTIN_object)
                {
                        emitCoerceToNumeric(imm30);
                        retType = OBJECT_TYPE;
                        already_coerced = true;
                }
                else if(bt == BUILTIN_float || bt == BUILTIN_float4)
                        retType = v.traits;
                else
                        retType = NUMBER_TYPE;
        }
#endif // VMCFG_FLOAT
        if(!already_coerced)
                emitCoerce(retType, imm30);
        coder->write(state, pc, opcode, retType);
        break;
}
```

不会执行

会执行

问题

- *emitCoerce(retType, imm30)* (行1302) 直接使用变量 *imm30* 而没有做边界检查
  - 事实上，*imm30* 即是 opcode declocal / inclocal 的参数
  - 当函数 *emitCoerce()* 可以被输入任意的 *imm30* 值的时候，多种情况的代码执行情景会发生
- 事实上，上面的代码片段里有一个典型的 checkLocal() 函数，这个函数本来就是设计用于边界检查的

```
1287            FrameValue& v = checkLocal(imm30);
```

- 然而，它却被这个 *VMCFG_FLOAT* 宏给禁用了.

```
case OP_inclocal:
case OP_declocal:
{
        Traits* retType = NUMBER_TYPE;
        bool already_coerced = false;
#ifdef VMCFG_FLOAT
         if(pool->hasFloatSupport())
        {
                FrameValue& v = checkLocal(imm30);
                BuiltinType bt = Traits::getBuiltinType(v.traits);
                if(bt == BUILTIN_none || bt == BUILTIN_any || bt == BUILTIN_object)
                {
                        emitCoerceToNumeric(imm30);
                        retType = OBJECT_TYPE;
                        already_coerced = true;
                }
                else if(bt == BUILTIN_float || bt == BUILTIN_float4)
                        retType = v.traits;
                else
                        retType = NUMBER_TYPE;
        }
#endif // VMCFG_FLOAT
        if(!already_coerced)
                emitCoerce(retType, imm30);
        coder->write(state, pc, opcode, retType);
        break;
}
```

不会执行

会执行

思考

- 现在，这个漏洞成因已经很明显了。漏洞存在于AVM2函数验证过程的核心部分
- 因此，任意的Flash版本 (Win8, Chrome) 都受影响
- 触发这个漏洞真的 *很容易*：
  - 只要给一个相对大点的参数给随便一个 *declocal* 或 *inclocal* opcode.
- 因此，问题来了：为什么如此一个简单的漏洞直到不久前才被发现呢？这难道不是一个有趣的问题吗？☺
  - 我们都知道，每天有很多的研究员在fuzzing Flash
  - 再加上无数的Google机器*。。

[* http://googleonlinesecurity.blogspot.in/2011/08/fuzzing-at-scale.html](http://googleonlinesecurity.blogspot.in/2011/08/fuzzing-at-scale.html)

在修订历史中挖掘

- 一个有理由的猜测是，这个漏洞被引入到Flash Player中没多久
- Tamarin-redux 源代码同样记录了所有的历史上的代码修改
- 让我们来找出它！

tamarin-redux - diff - core/Verifier.cpp

summary | shortlog | changelog | graph | tags | files | changeset | file | revisions | annotate | diff | raw

core/Verifier.cpp

changeset 6902	9e138314d408
parent 6895	f7966eef4b33
child 6924	78e6caa24f45

```
--- a/core/Verifier.cpp
+++ b/core/Verifier.cpp
@@ -1282,21 +1282,21 @@ namespace avmplus
                     state->setType(imm30, NULL, false);
                     break;
                 }

                 case OP_inclocal:
                 case OP_declocal:
                 {
                     Traits* retType = NUMBER_TYPE;
-                    FrameValue& v = checkLocal(imm30);
                     bool already_coerced = false;
 #ifdef VMCFG_FLOAT
                     if(pool->hasFloatSupport())
                     {
+                        FrameValue& v = checkLocal(imm30);
                         Traits* vt = v.traits;
                         if(!vt || !vt->isNumeric())
                         {
                             emitCoerceToNumeric(imm30);
                             retType = OBJECT_TYPE;
                             already_coerced = true;
                         }
                         else if(vt == FLOAT_TYPE || vt == FLOAT4_TYPE)
```

checkLocal(imm30) 正在被从这个分支的开头转移到这个宏所控制的代码块，这个宏控制的代码块并不会真的执行到！

实际上这个“转移”引入了这个安全漏洞！

修订 9e138314d408

- *"author Virgil Palanciuc <virgilp@adobe.com> Wed Nov 09 18:44:25 2011 +0200 (at Wed Nov 09 18:44:25 2011 +0200)“*

- **2011年11月9号**，这个漏洞被引入到Tamarin项目中

- 实验测试显示，这个漏洞被引入到正式的Flash Player中是从 **2012年6月8号** 发行的 11.3.300.257 版本开始的
  - 11.3.300.257 是第一个公开发行的11.3版本

一些有意思的结论

- **7个月**后，Adobe公司重新编译了 Tamarin 源代码到发行的Flash Player中

- 每当发行“第二大版本号”的Flash Player版本时，Adobe会重新集成更新的Tamarin源代码
  - **11.3.300.257**是第一个**11.3**版本
  - 虽然，这个结论还需要有关方面证实。。

- 因此，人们可以监视并检视Tamarin源代码而从中发现安全漏洞。然后等待下一个“第二大版本号”的Flash发行，来达到获得真正Flash漏洞的目的

Agenda

1	回顾Flash威胁
2	Flash ActionScript 3 架构
3	Flash JIT和减轻技术
4	案例: 利用Tamarin源代码来分析Flash漏洞
5	未来Flash安全的思考

当前Flash安全情景

- 消灭Flash漏洞：
  - 企业级的fuzzing每天都在进行 (Google/Adobe/MS guys).
  - MS/Google的产品现在集成了Flash Player (Win8/PepperFlash) 因此他们正在积极协助改进Flash安全

- 减轻Flash利用：
  - JIT加固(已讨论)
  - 沙盒化Flash (Chrome上的PepperFlash).
    - 沙盒化同样有助于减轻Flash JIT 攻击因为每个Flash内容都在不同的进程里被处理

未来 – 漏洞发掘

- 我们将持续看到源自文件格式的漏洞：
  - 字体 / 图像 / 视频
  - AMF(Action Message Format)

- AVM虚拟机级别的漏洞已经减少了 (thanks, 因为它们通常都很具威胁性)

- Flash AS3 本地API (public class)
  - 在Flash ActionScript 3 本地API执行过程中发生的漏洞
  - 基于文件格式的fuzzing无法发现此类漏洞，因为这些漏洞在触发于脚本级别。
    - 好比你发掘IE DHTML漏洞一个道理
  - 可能会成为Flash漏洞的多产地，很多的API需要检视 (hmm, 他们不是开源的)

Reference Fuzzing!

```
08:15:45 PM:all_method_names:
08:15:46 PM:dispatchEvent
08:15:46 PM:hasEventListener
08:15:46 PM:willTrigger
08:15:46 PM:toString
08:15:46 PM:setRequestedUpdateInterval
08:15:46 PM:addEventListener
08:15:46 PM:removeEventListener
08:15:46 PM:method_count:7
08:15:46 PM:Fuzzing on <willTrigger>
08:15:46 PM:param_set: 1 from [1,1]
08:15:46 PM:Calling willTrigger on [object Accelerometer]
08:15:46 PM:*****Refs_Pool.length = 100*****
08:15:46 PM:*****Fuzzing in Reference Pool*****
08:15:46 PM:One Object test out
```

- Credit to Tavis Ormandy's *ReferenceFuzzer*.

未来 – 漏洞利用

- JIT加固是很强，但是还不够
  - 一些全新的发现将在不久后发布 :P

- Flash Player上的定制的堆管理器无疑是个问题，在某些特殊场合，它会对高级漏洞开发大大有用
  - 根本没有任何加固技术，还很原始很脆弱

- 任何时候，检视AVM2过程中的深入特性都是有价值的
  - 这些特性可能有助于高级漏洞利用
  - 一个实际的例子就是针对AS3类型混乱漏洞的利用研究，发布于2011年的CanSecWest上

McAfee
An Intel Company

Questions?

Haifei_Li@McAfee.com